Distributed Systems

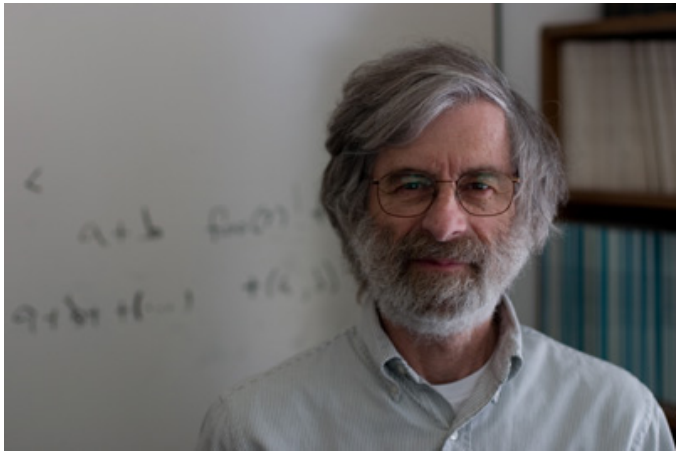
Time Clocks and Ordering of events
Distributed Snapshots – Determining Global States

Dinesh Bhat - Advanced Systems
(Some slides from 2009 class)
CS 6410 – Fall 2010

Time, Clocks, and the Ordering of Events in a distributed System

Leslie Lamport

Author



- Leslie Lamport
  - B.S., M.I.T., 1960. (Maths)
  - Ph D, Brandein University, 1972

  - ACM SIGOPS Hall of Fame Award (2007)
  - IEEE John von Neumann Medal (2008)
  - Microsoft Research (from 2001-present)

Outline

- ❑Introduction
- ❑Partial Ordering of Events
- ❑Logical Clocks and Total ordering
- ❑Distributed algorithm: An Example
- ❑Strong Clock Condition using Physical Clocks
- ❑Discussion

Introduction

- Defn 1 : Collection of nodes connected via network representing a single coherent system
- Defn 2 : Collection of processes within a computer

WHY DS ?

- Resource sharing/
  Load Balancing
- Computation speedup
- Reliability
- SW/HW preference
- Data access

Introduction

Design issues in building a DS:

- **Transparency** – the distributed system should appear as a conventional, centralized system to the user.
- **Fault tolerance** – the distributed system should continue to function in the face of failure.
- **Failure detection –** in case of asynchronous DS, its hard to differentiate between message delay Vs lost message
- **Scalability** – as demands increase, the system should easily accept the addition of new resources to accommodate the increased demand.

Introduction

Complexities in DS implementation :

- Unpredictable order of events in a distributed system

  Problem 1- Update A needs to occur before update B

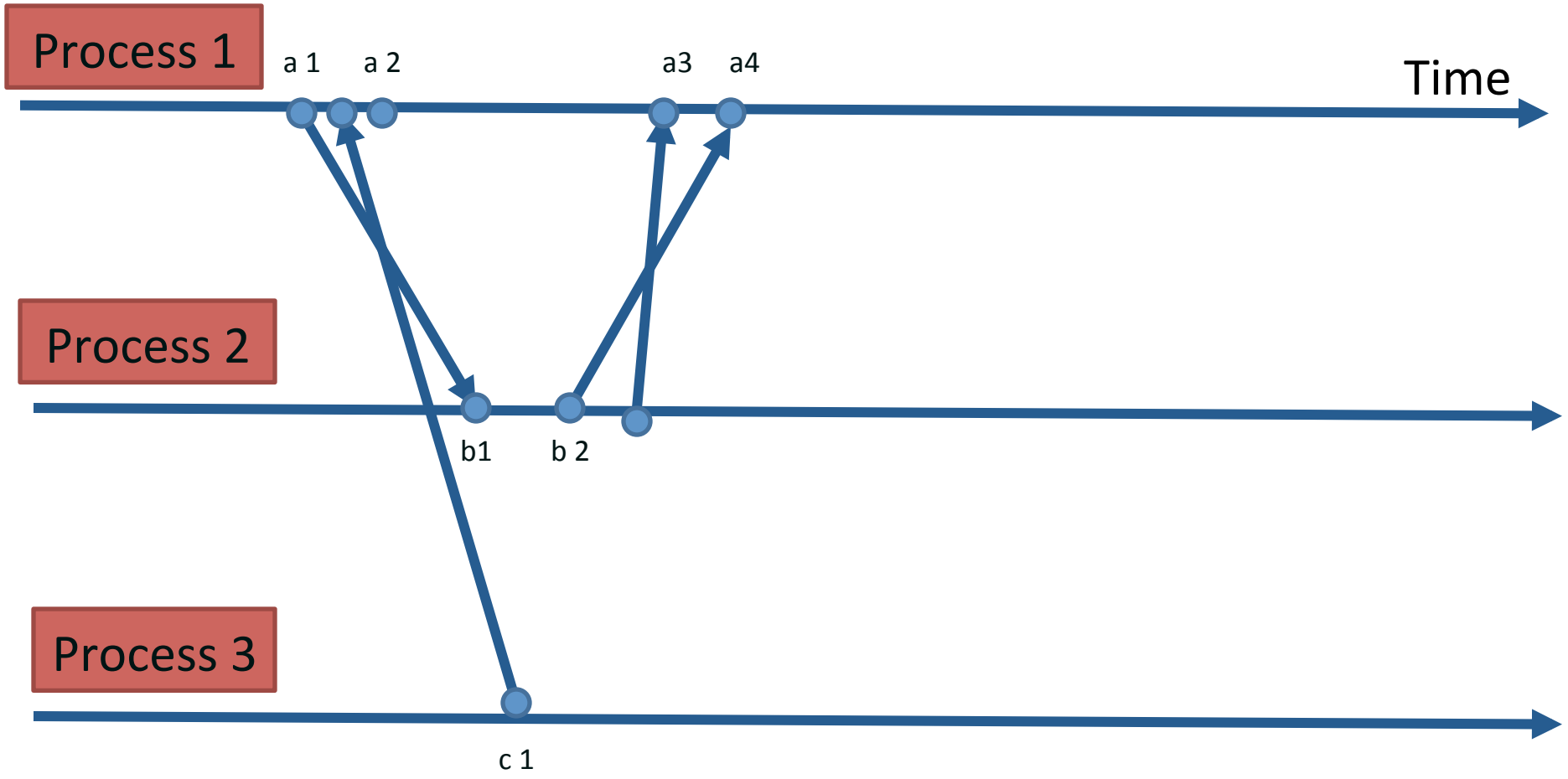
- No global clock shared by processes – if there was any, FCFS could have solved all issues of event ordering

Some ordering could be implicit

- Events in a single process happen in order
- Message between processes must be sent before they are received

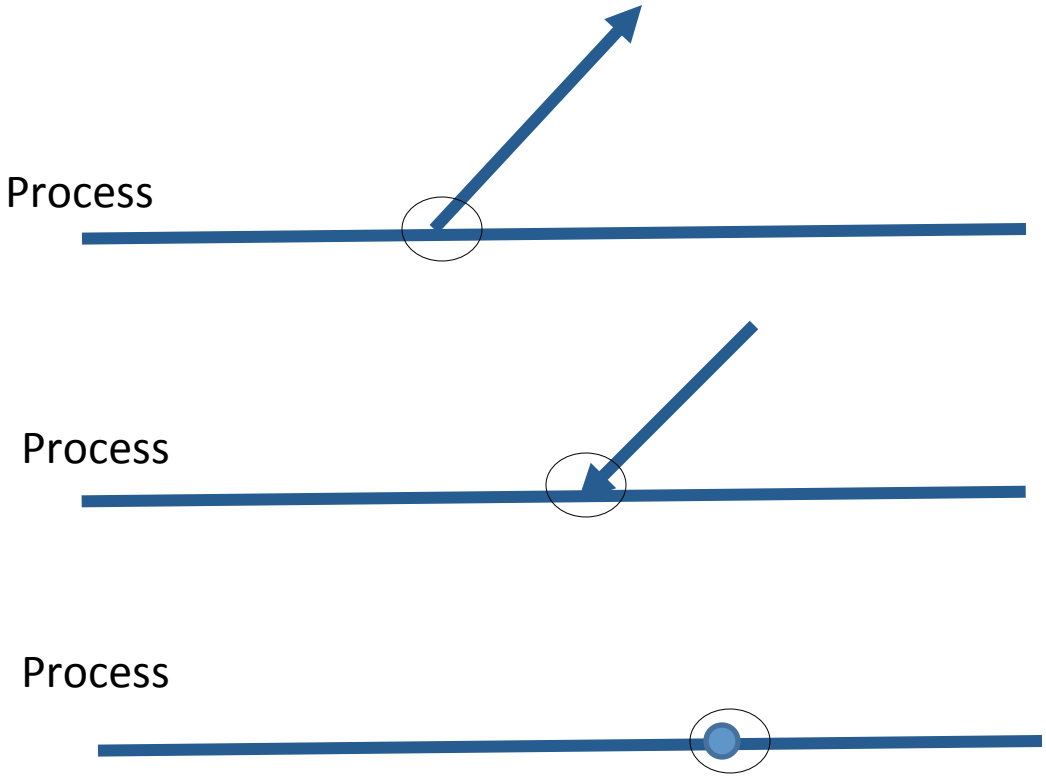
WHAT IS THE SOLUTION ???

Space Time diagram of events



Events

- Send Event
- Receive Event
- Local event



Partial Ordering of Events

- Causal events satisfying relation → "Happens Before"

  a) Two local events of a process : a → b

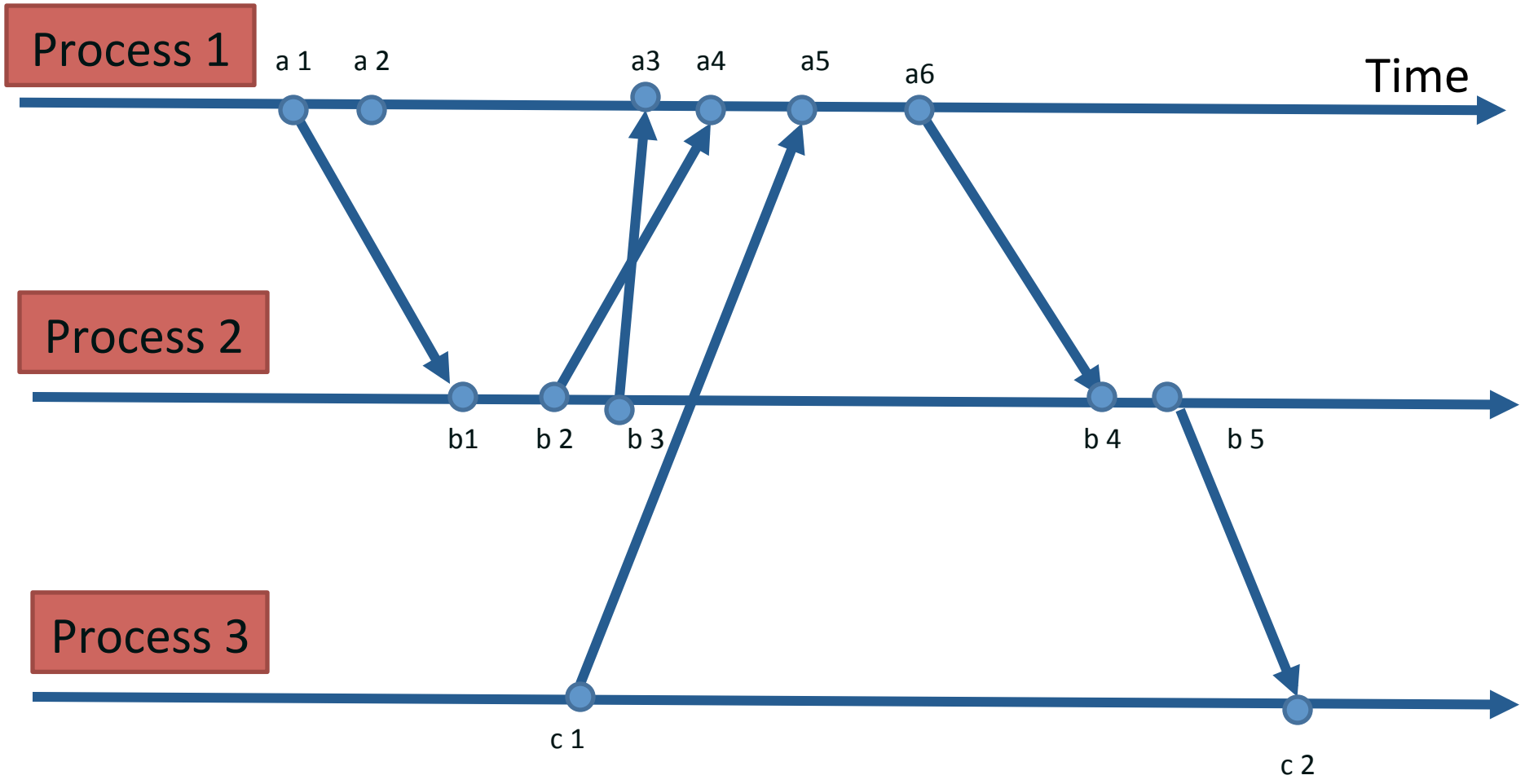
  b) Two endpoints of a same message : a→b

  c) Transitive : If a→b and b→c, then a→c

- Concurrent events

  a ↛ b, a and b run independently

Space Time diagram of events



Logical Clocks to implement Partial Ordering of events

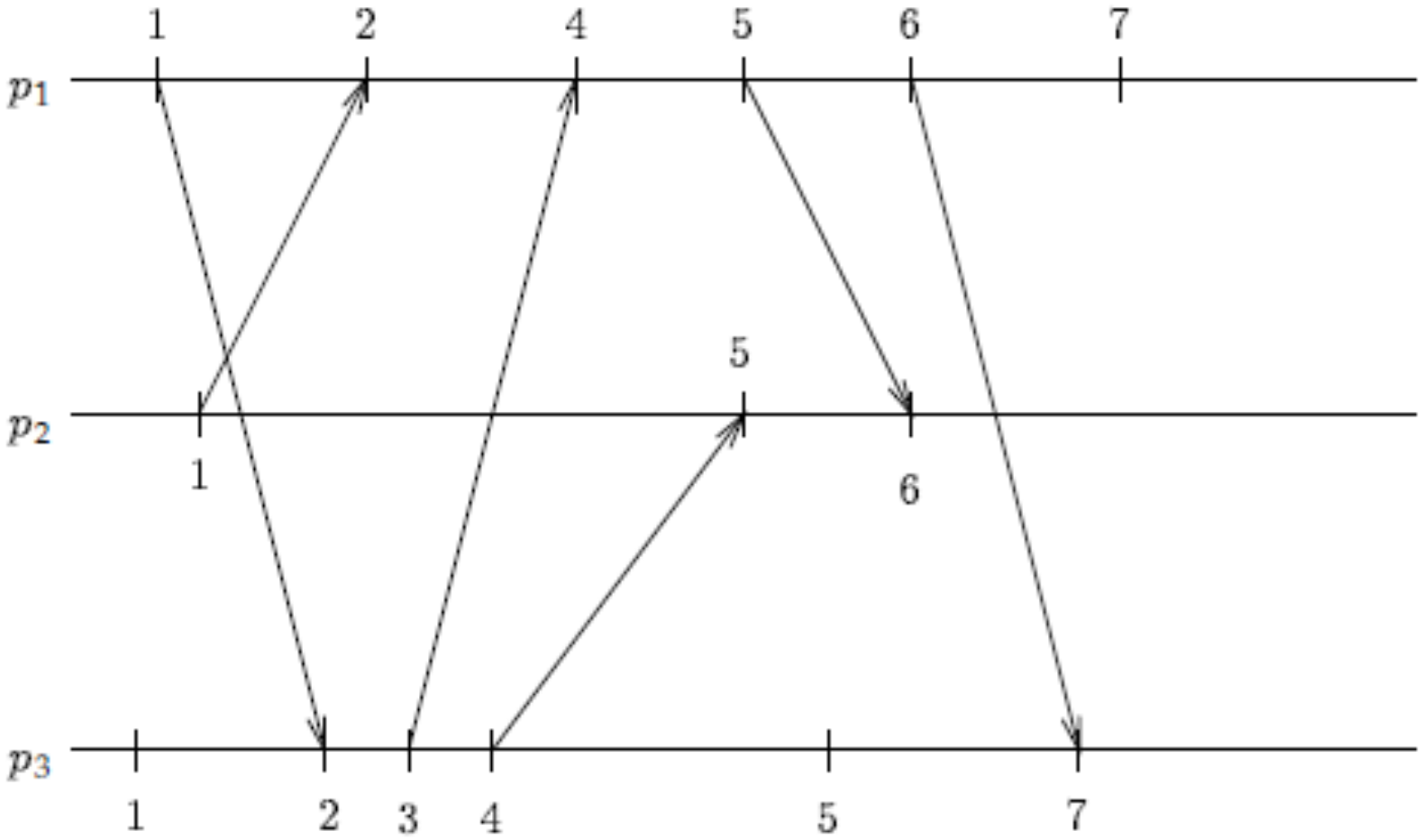
- Logical Clock = Way of assigning a number to an event

- Following Clock condition should be satisfied for partial ordering :

  – If a → b, then C(a) < C(b)

  "Converse is not true"

Logical Clocks to implement Partial Ordering of events



Logical Clocks(Contd..)

- ‘a’ and ‘b’ are local events, to hold C(a) < C(b),
  - Assign distinct numbers to every successive local events in incremental fashion
- ‘a’ and ‘b’ are send and receive events of message M, to hold C(a) < C(b),

$$LC(e_i) := \begin{cases} LC + 1 & \text{if } e_i \text{ is an internal or send event} \\ \max\{LC, TS(m)\} + 1 & \text{if } e_i = receive(m) \end{cases}$$

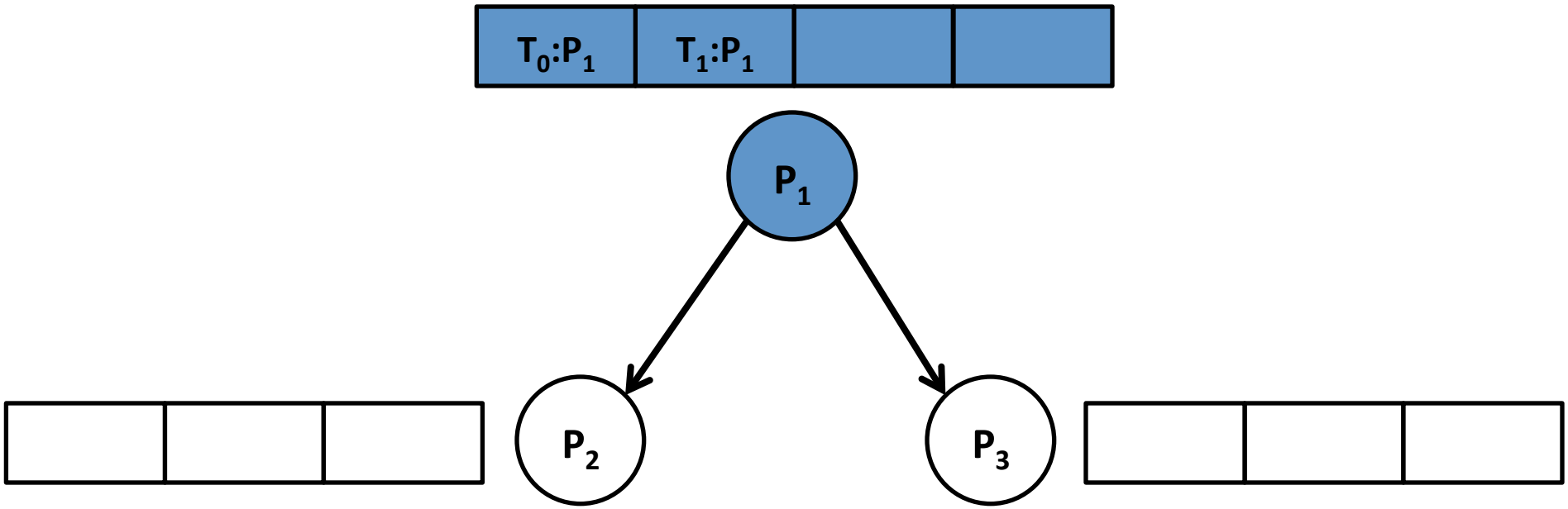
Total ordering of events

- Useful in implementing a distributed system
- Logical clocks could be extended to obtain total ordering
- If C(a) == C(b) && P(a) < P(b), then a => b

- Example : mutual exclusion problem
  - Multiple processes contending for one resource

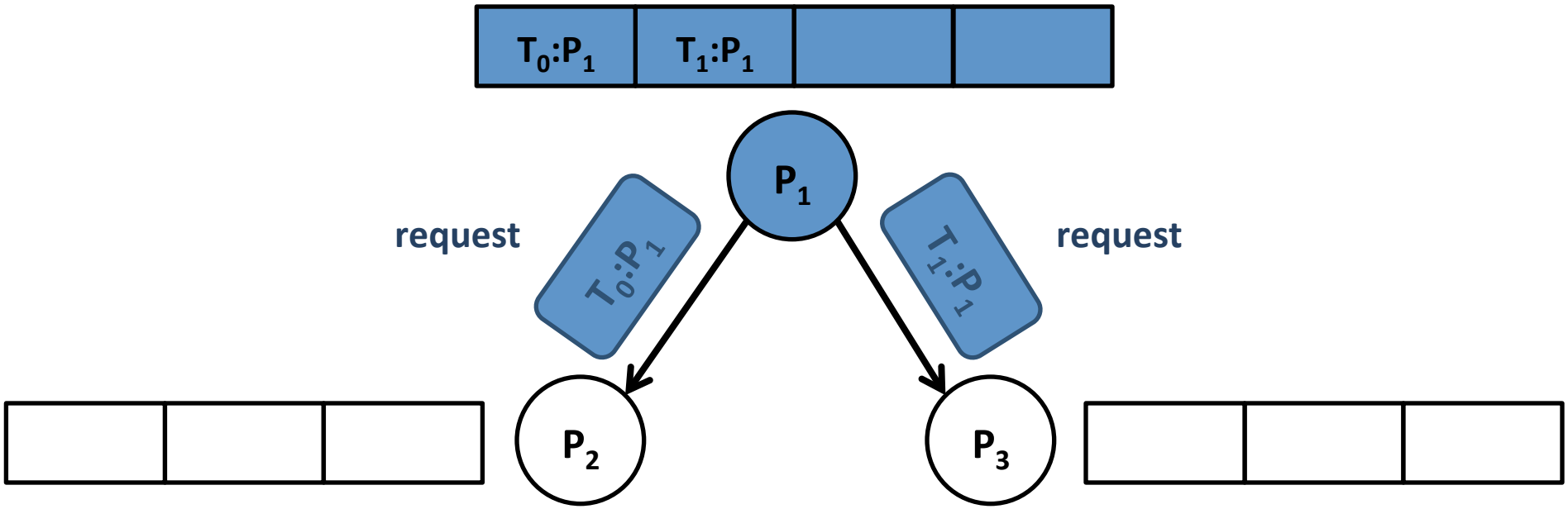
Total Ordering of Events : Example

- **Step 1: P_i Sends Request Resource**
  - P_i sends **Request T_m:P_i** to P_j
  - P_i puts **Request T_m:P_i** on its request queue



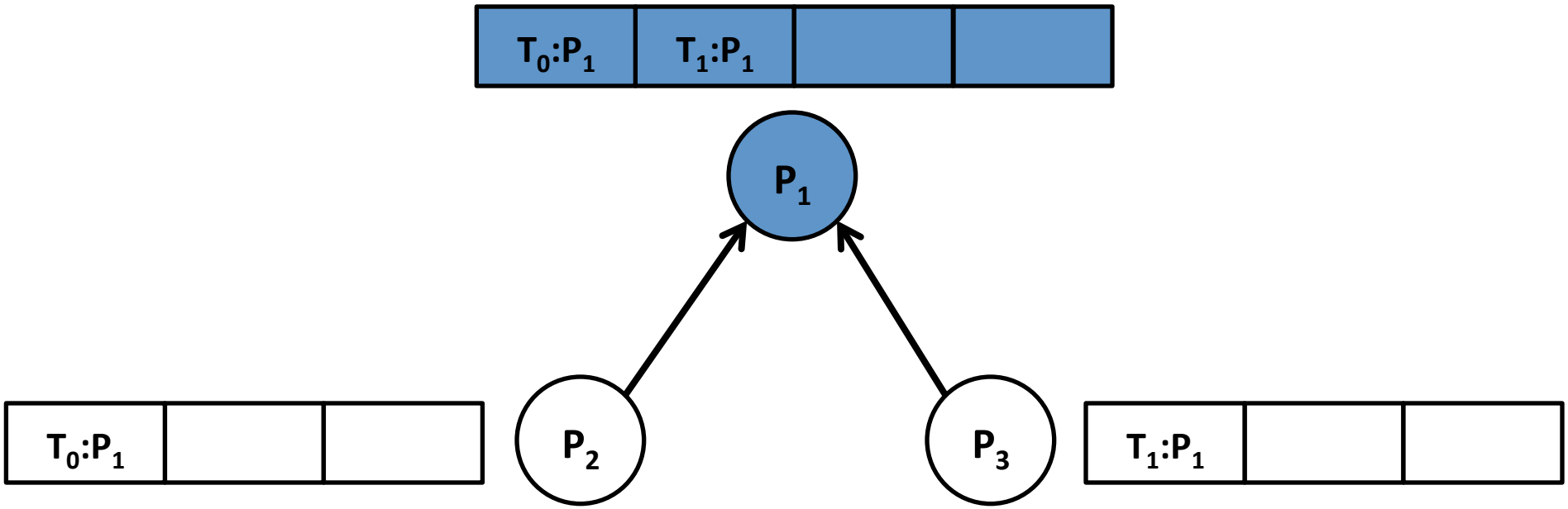
Total Ordering of Events : Example

- **Step 1: P_i Sends Request Resource**
  - P_i sends **Request T_m:P_i** to P_j
  - P_i puts **Request T_m:P_i** on its request queue



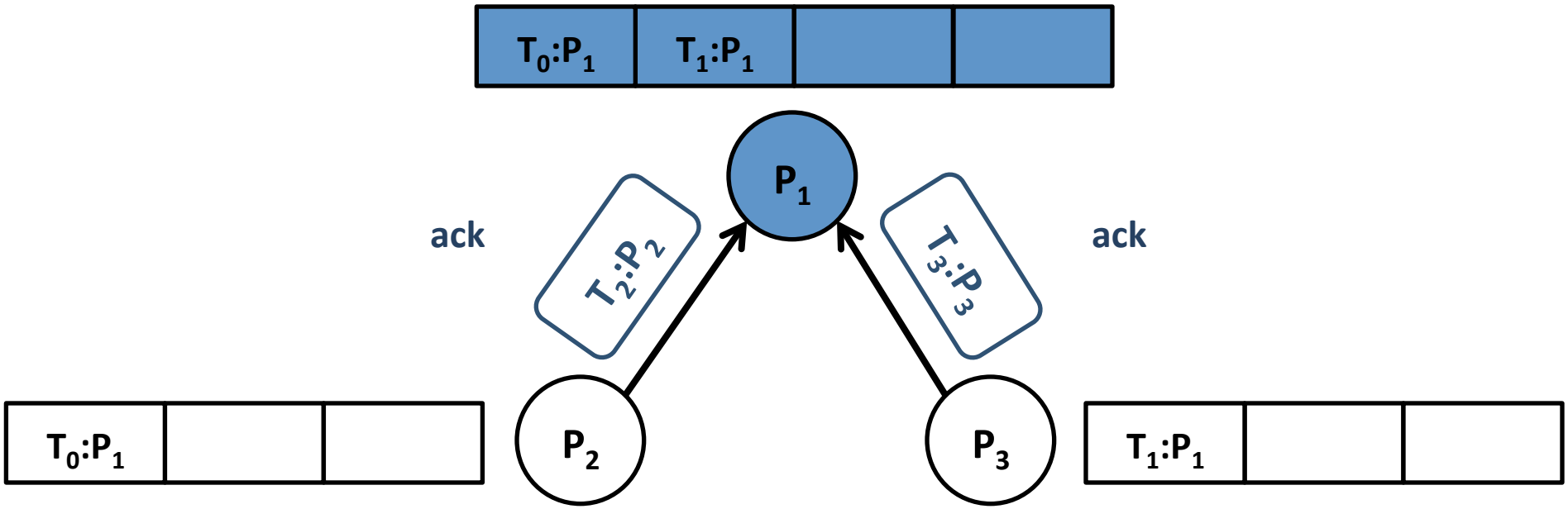
Total Ordering of Events : Example

- **Step 2: P_j Adds Message**
  - P_j puts **Request T_m:P_i** on its request queue
  - P_j sends **Acknowledgement T_m:P_j** to **P_i**



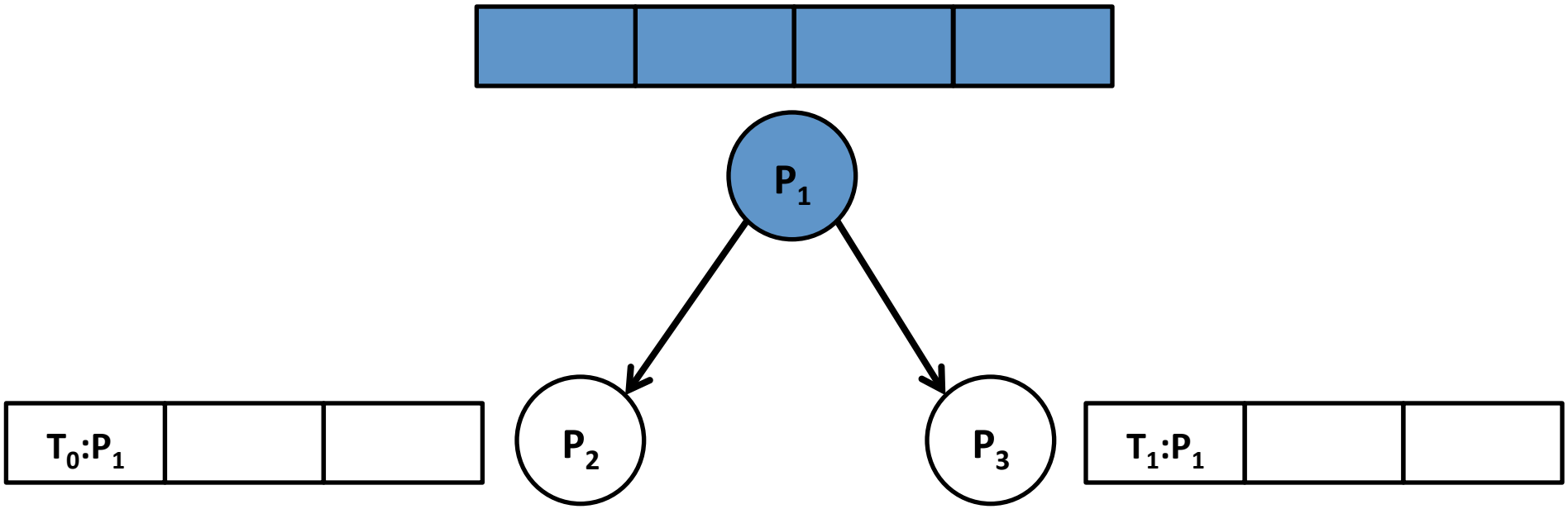
Total Ordering of Events : Example

- **Step 2: P_j Adds Message**
  - P_j puts **Request T_m:P_i** on its request queue
  - P_j sends **Acknowledgement T_m:P_j** to **P_i**



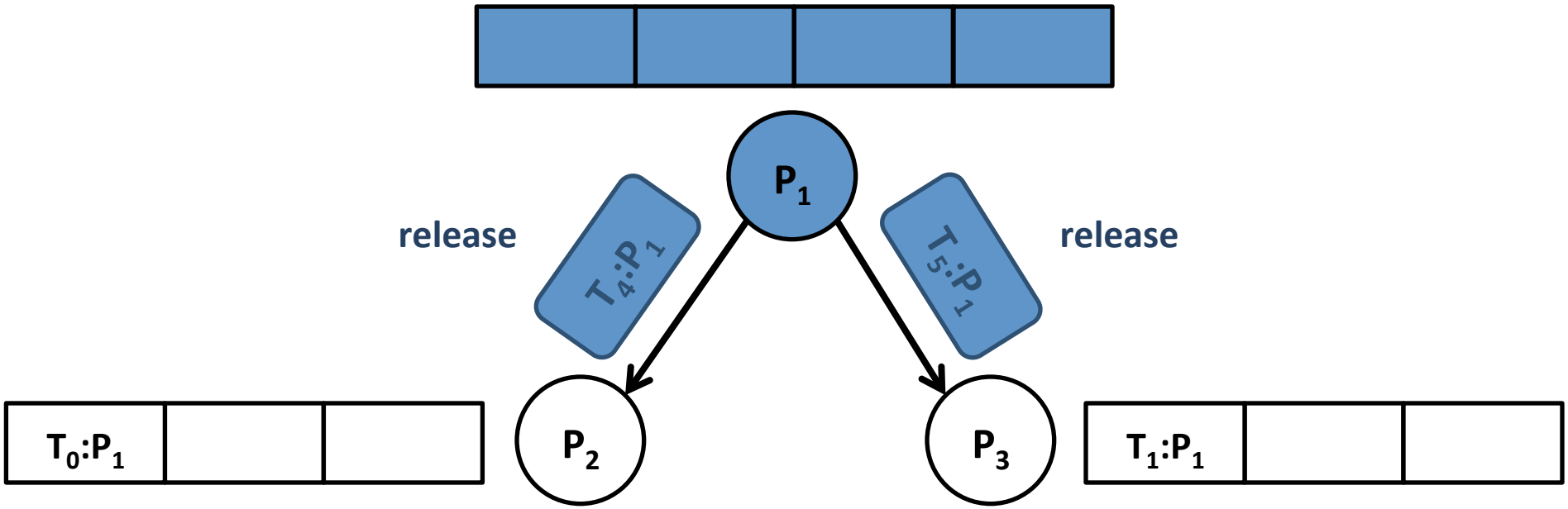
Total Ordering of Events : Example

- **Step 3: P_i Sends Release Resource**
  - P_i removes **Request T_m:P_i** from request queue
  - P_i sends **Release T_m:P_i** to each P_j



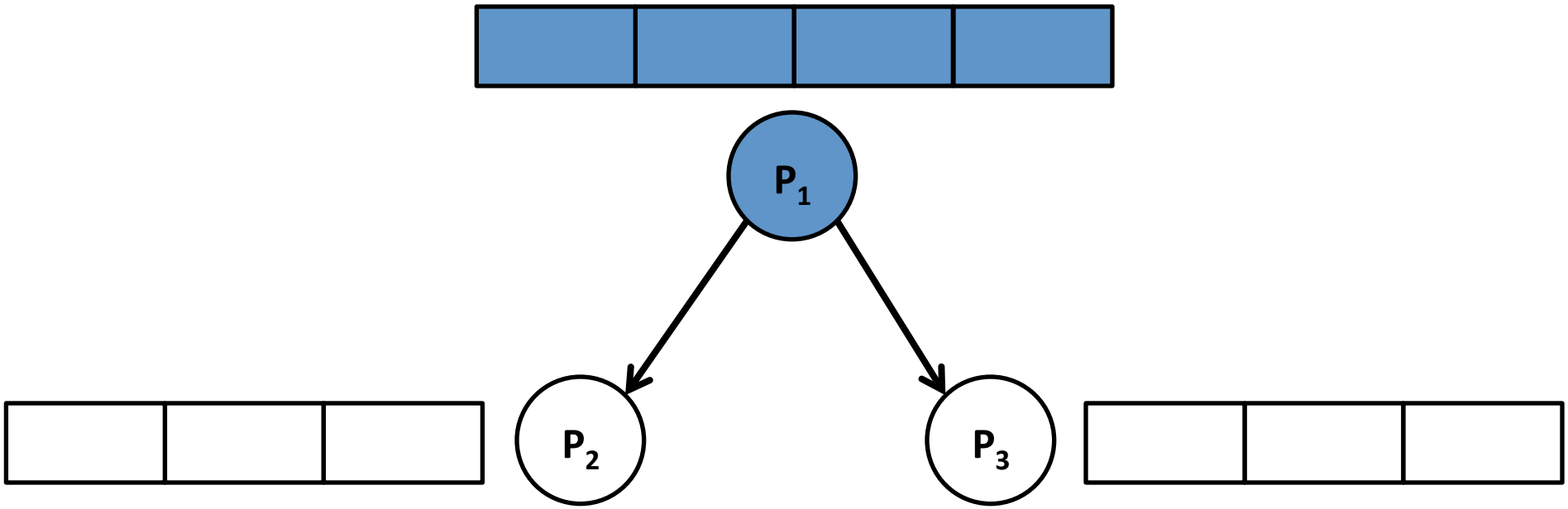
Total Ordering of Events : Example

- **Step 3: P_i Sends Release Resource**
  - P_i removes **Request T_m:P_i** from request queue
  - P_i sends **Release T_m:P_i** to each P_j

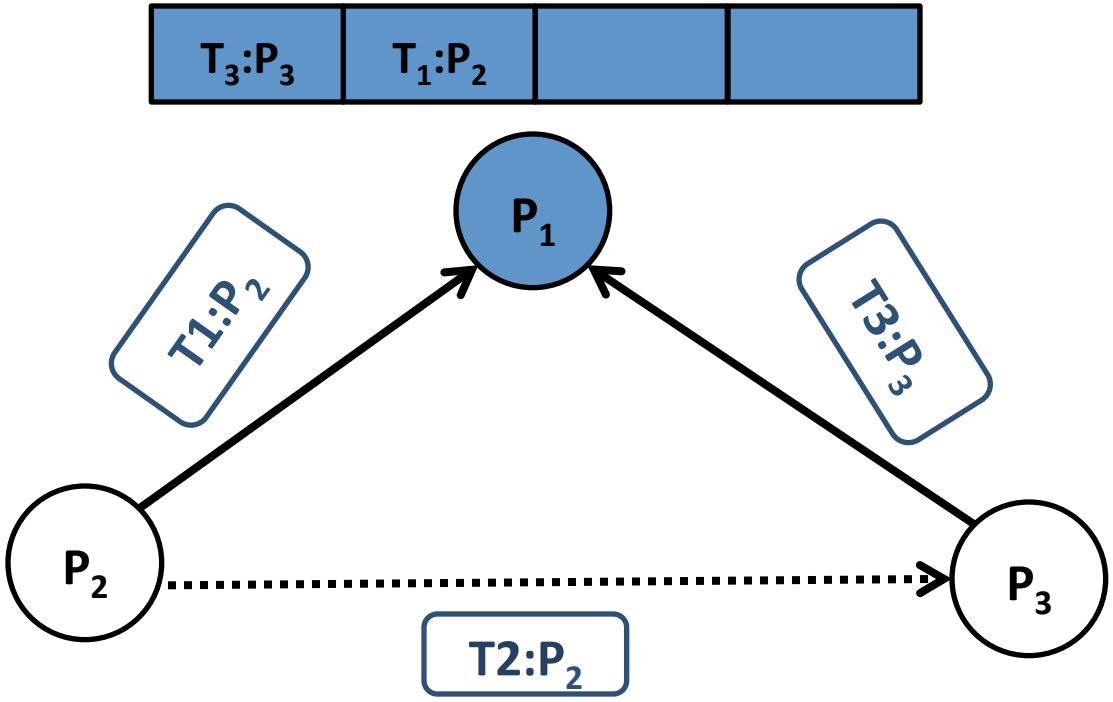


Total Ordering of Events : Example

- **Step 4: P_j Removes Message**
  - P_j receives **Release T_m:P_i** from P_i
  - P_j removes **Request T_m:P_i** from request queue



Anomalous behavior



Solution 1 :

Attach a timestamp with T2:P2

Solution 2 : Physical Clocks

- Board diagram

Discussion

- Proof of concept
- Defines event ordering in distributed systems
- Building systems on top of Specifications, Set of assumptions (No node failures, reliable channels)
- Logical Clock counters – overflow issues ?
- Set of assumptions were strong ?

K. Mani Chandy and Leslie Lamport

DISTRIBUTED SNAPSHOTS

Authors

- Leslie Lamport
- K Mani Chandy
  - Simon Ramo Professor, CalTech (since 1987)
  - Ph D, MIT, 1969
  - Honeywell, IBM, UT Austin
  - IEEE Koji Kobayashi Award (1987)
  - Event driven architecture(marketed by Avaya)

Outline

- ❑ Motivation
- ❑ Cuts and Consistent cuts
- ❑ Distributed system model
- ❑ Distributed Snapshots
- ❑ Global State Detection Algorithm
- ❑ Properties of recorded Global state
- ❑ Stability Detection

Motivation and Goals

- A process in a DS determines the global state of the system which is useful in detecting stability of the system
- Challenges :
  - Communication delay
  - Relative speeds of computations differ

- Stable properties examples:
  - Computation is terminated
  - kth computational phase is terminated
  - System is deadlocked

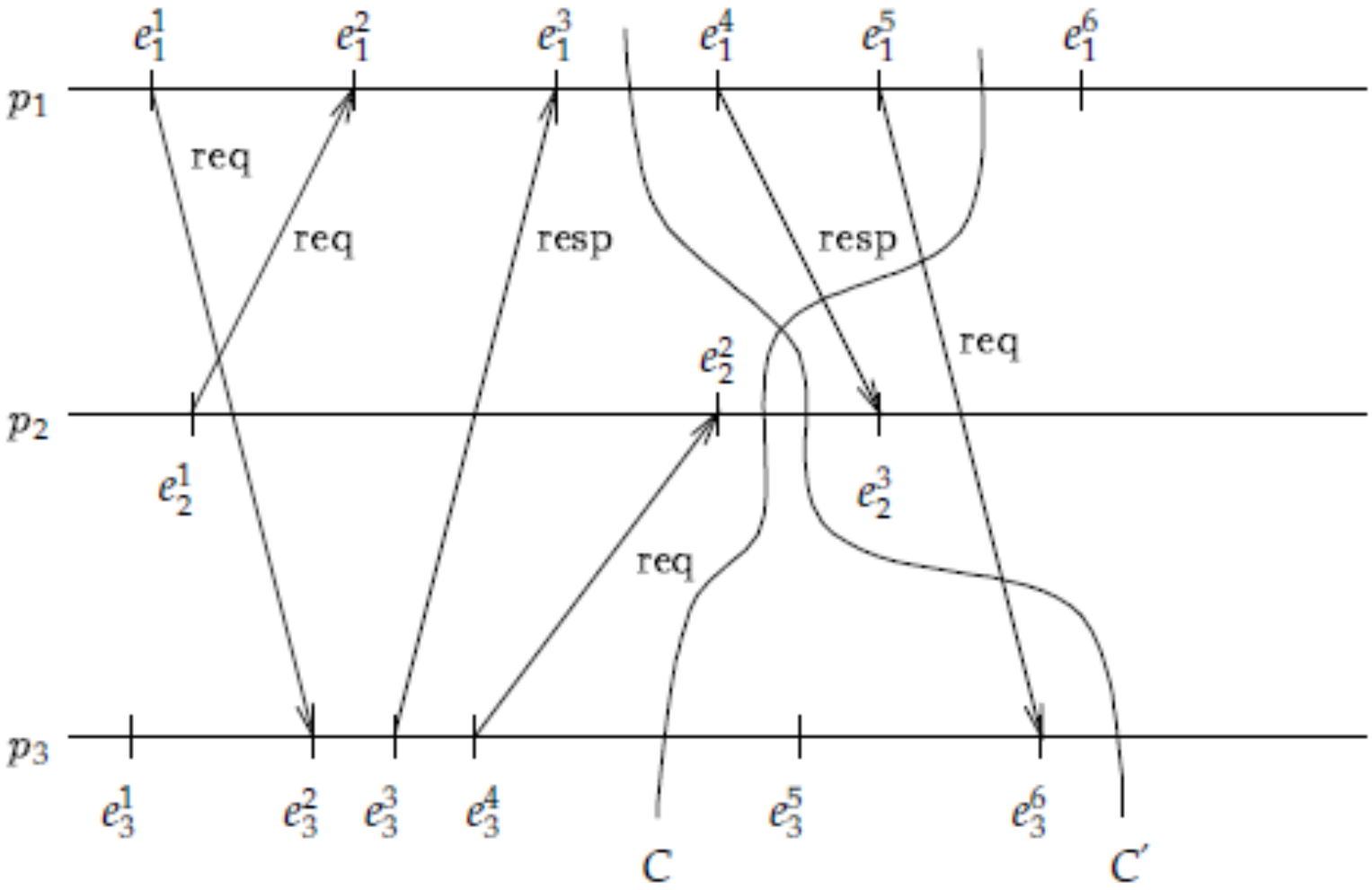
- State detection algorithm is similar to capturing panoramic view of migrating birds
  - Composite picture should be meaningful
  - Moving birds add complexity to the process

Distributed System Model

- State of processes and Channels
  - States of processes and Channels are defined by events and messages respectively

- Event e = <p, s, s', M, c>
  - process p
  - s and s' are states
  - channel c and message M

Cuts and Consistent Cuts



Consistent Global State

- A cut C is consistent if for all events e and e’

$$\left(e \in C\right) \wedge \left(e' \rightarrow e\right) \Rightarrow e' \in C$$

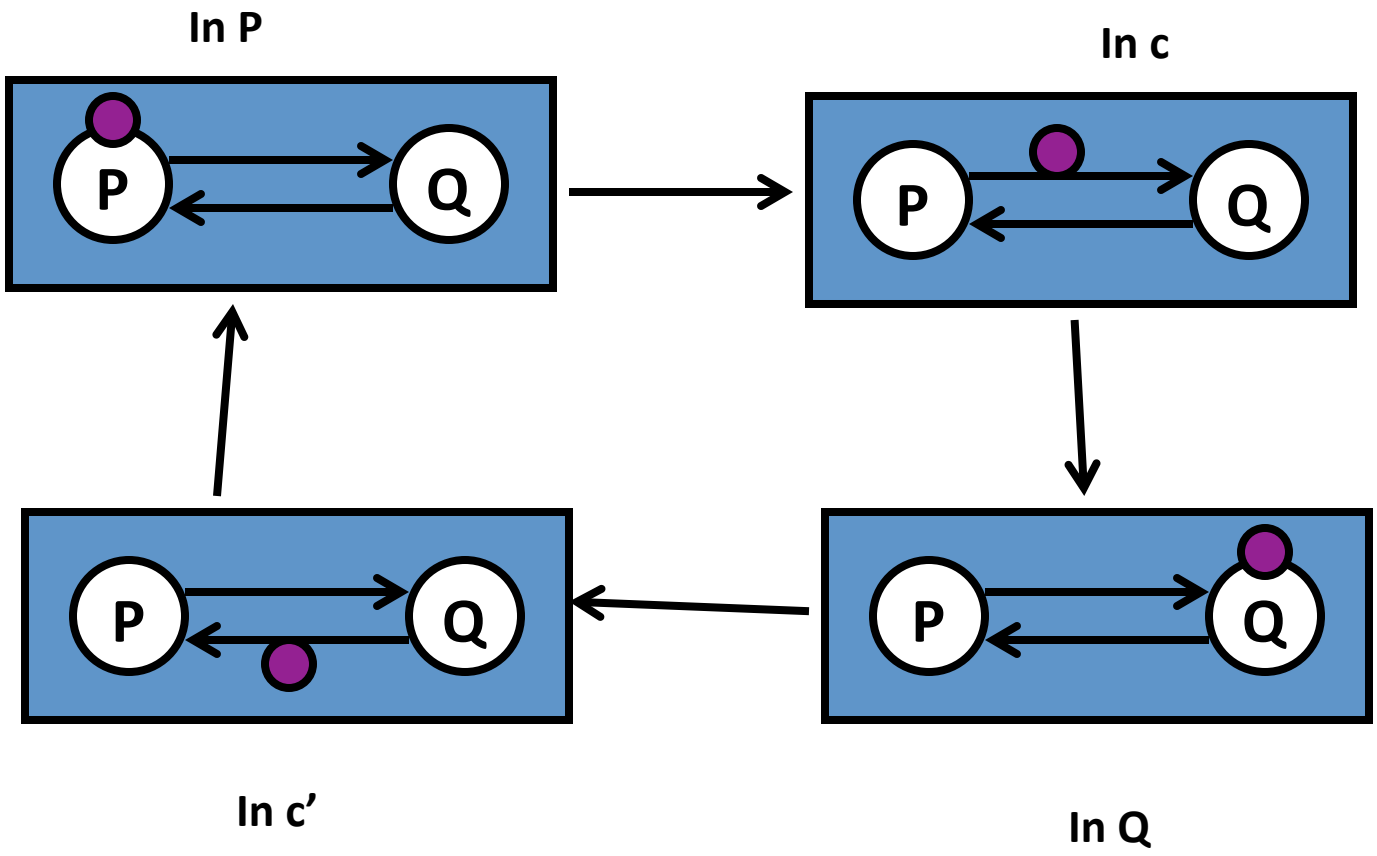
- Intuitively if an event is part of a cut then all events that happened before it must also be part of the cut
- A consistent cut defines a consistent global state

Assumptions

- Processes do not fail
- Reliable communication channels
- FIFO delivery between a pair of processes
- Channels have infinite buffers

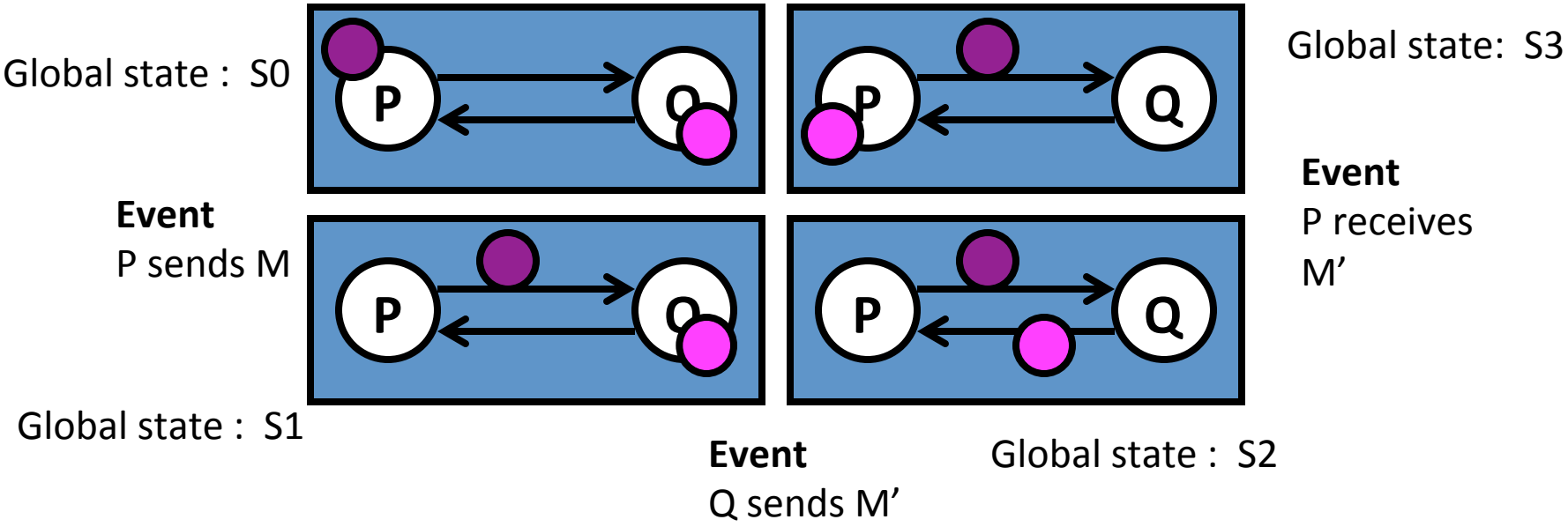
Distributed System Model

- **Single Token System – compute global states**



Distributed System Model

- **Nondeterministic System**



Global State Detection Algorithm

- **Marker Snapshot Protocol**
  - *P* records its state and pushes an empty marker M on all outgoing channels
  - *Q,* when receives a marker M along its incoming channel c,
    - If it was not in recording state,
      - Start recording Q's state
      - Record c's state as empty state
      - Pushes the Marker M onto all its outgoing channels
    - If it was already in recording state,
      - Stops recording on incoming channel c, and records the state of c as the sequence of messages received since Q started recording and before Marker M is received on this channel

When Q receives markers on all its incoming channels, stop recording the state of Q and its incoming channels and 'call it a day' for *Q*.

Properties of recorded global state

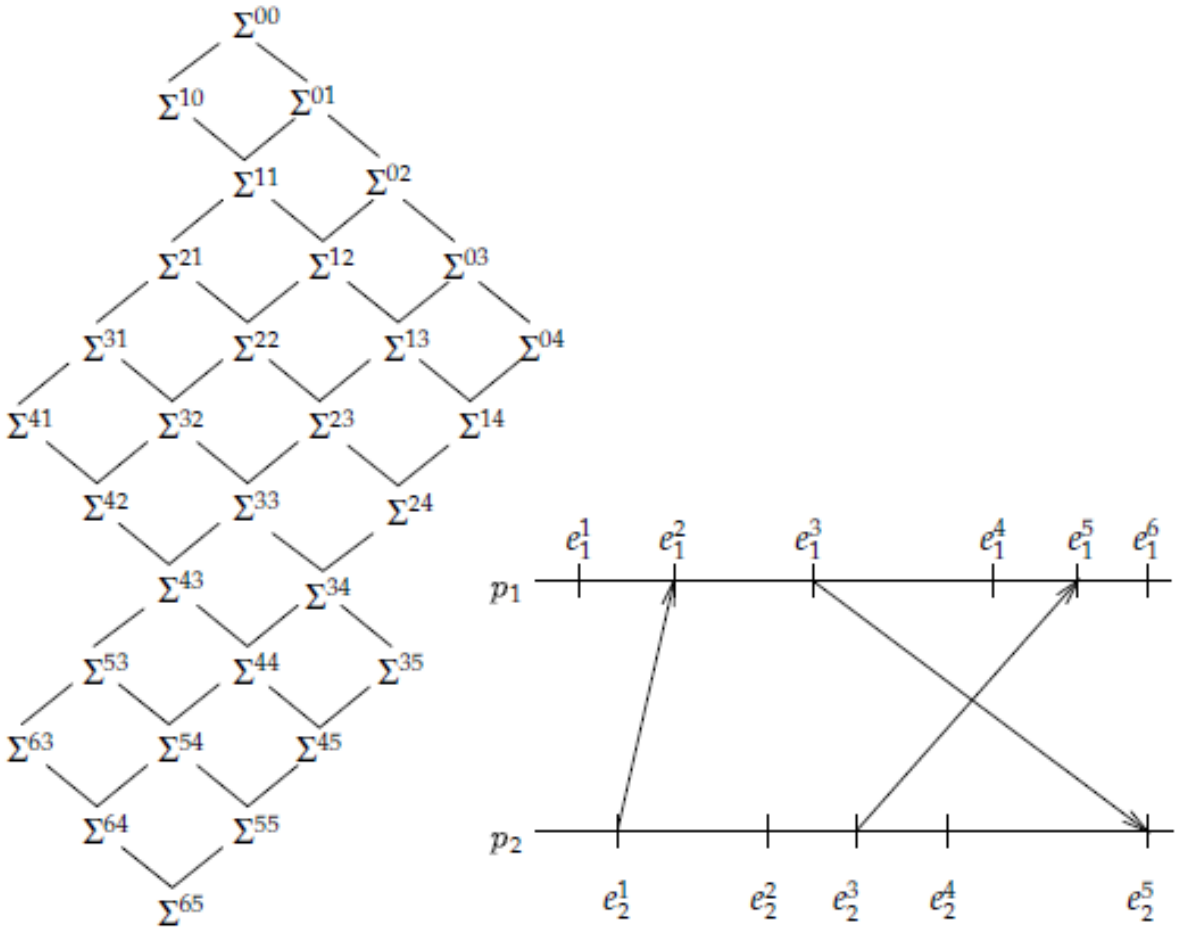


Figure 3. A Distributed Computation and the Lattice of its Global States

Stability Detection

- **Algorithm**
  - **Start:** definite=false, $y(S_0)$=false
  - **Repeat:** record S*, definite=y(S*)
- **Implications of "definite"**
  - **definite == false:** can not say YES/NO stability
  - **definite == true:** stable property at termination
- **Correctness**
  - S_0 can lead to S*, S* can lead to S_t
  - for all j: $y(S_j) = y(S_j+1)$

Discussion

???

Other References

- WikiPedia
- http://www.cs.cornell.edu/courses/cs4410/2008fa/
- Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms